A library search prototype using Ontology

Defined using Protégé

April 2023

Version History:

No.	Date	Comments
1.0	28-March-2023	Outlined the structure
1.1	6-April-2023	Review and applications

1 Table of Contents

2 Introduction

Assuming the local library is an organisation that aims to improve its services to the community by being more efficient in its service delivery. The library has an extensive collection of books for adults and children, ranging from fiction to non-fiction. The library is facing the challenge of improving the search capabilities of its current system to help visitors find books more efficiently. The proposed ontology will allow the library to organise and structure its knowledge about books, authors, genres, subjects, and users meaningfully, which can be used to improve the search experience for visitors.

3 Designing in Protégé

The ontology design choices for the local library search engine were made based on several factors, including the library's collection of books, the needs of its users, and the overall goal of improving the search experience.
(Knublauch et. al., Medina et. al., Tom Gruber)

Here are some of the rationales for the ontology design choices:

1. Book (or Resource) class: This class has several properties: title, author, genre, subject, and publisher.
2. Author class: The Author class was created to represent the authors of the books in the library's collection.
3. Genre class: The Genre class was created to represent the different genres of books in the library's collection.
4. Location class: This class defines the book or resource availability in various locations.
5. User class: The User class was created to represent the library's users. This class has several properties: name, age, and borrowing history. These properties were chosen because they can personalise the search results based on the user's interests and preferences.
6. Publisher
7. Review
8. Bookmark

Here are the steps to design a prototype using Protégé software:

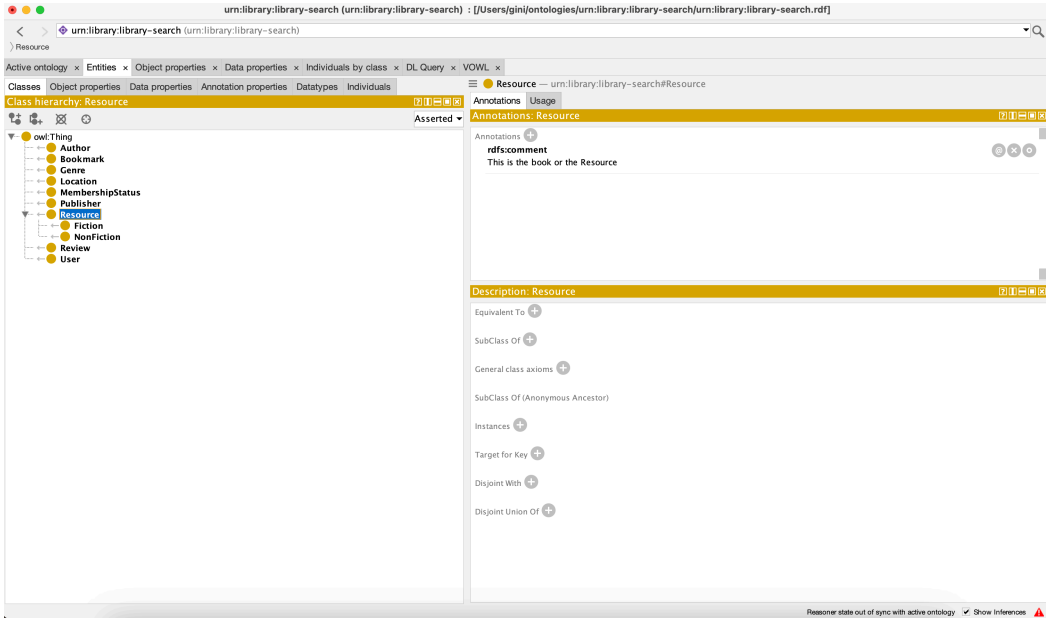
1. Define the domain: In this case, the domain is a local library with books for adults and children, both fiction and non-fiction.
2. Identify the main classes: The main courses in this domain could be Book, Author, Genre, Subject, and User.
3. Define the properties: The properties are the relationships between the classes. For example, a Book has an Author, a Genre, a Subject, and a User.

The Author writes the Book, the Genre categorises the Book, the Subject describes the content of the Book, and the User borrows the Book.

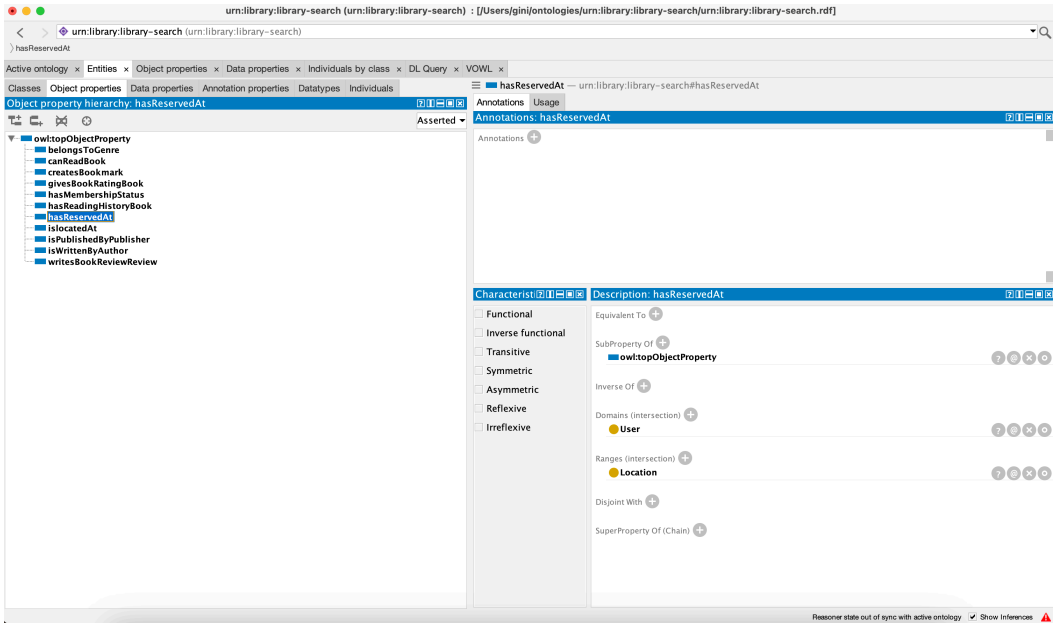
4. Create the hierarchy: Create an order of classes by grouping similar concepts. For example, under the Book class, you could create subclasses for Fiction and Non-Fiction.
5. Define the relationships: Define relationships between classes using object properties.
6. Add instances: Add instances of the classes to the ontology
7. Define constraints: Define constraints using restrictions. For example, you can define that a Fiction book cannot belong to the Non-Fiction class.
8. Reasoning: Use reasoning to infer new knowledge from ontology. For example, if a user searches for a book by a particular author, the search engine can infer other books written by the same author.
9. Test the ontology: Test the ontology using queries to ensure it works correctly.

Overall, the ontology design choices comprehensively represented the library's collection of books, authors, genres, subjects, and users. The design choices were based on the needs of the library's users and the goal of improving the search experience by providing personalized, relevant, and enriched search results.

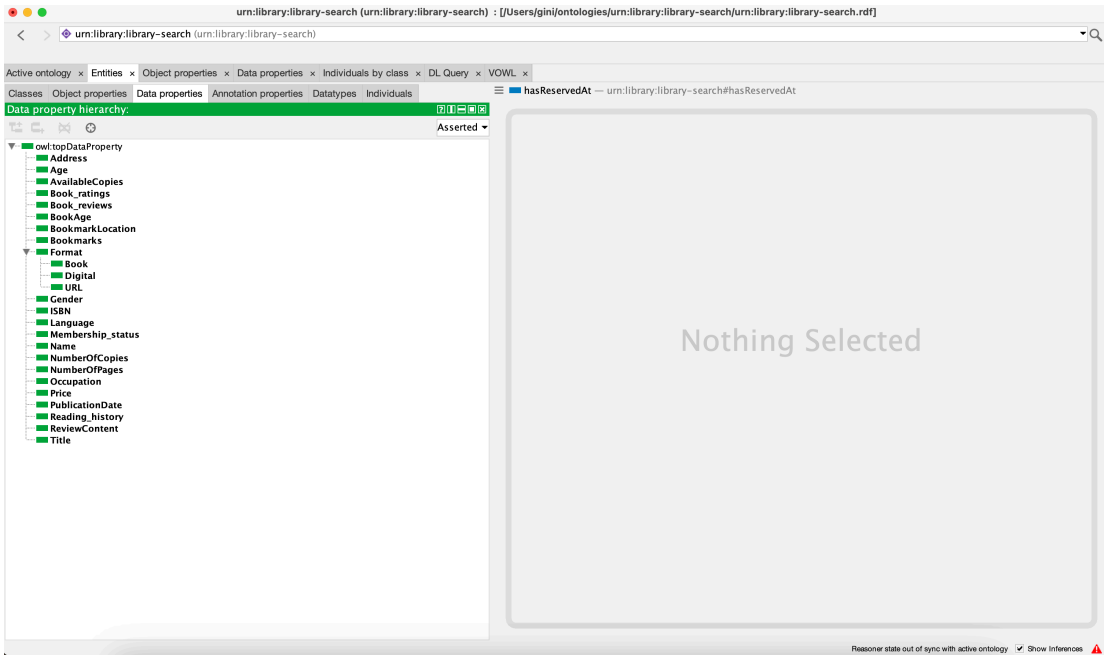
Classes



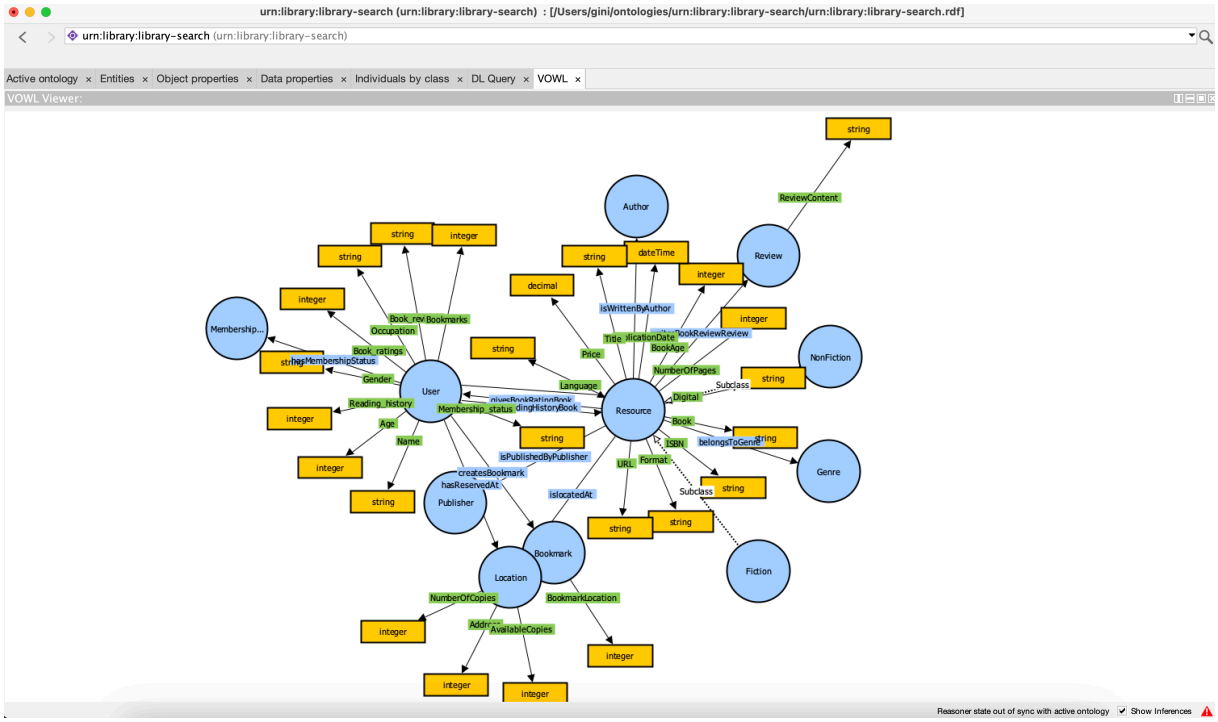
Object Properties



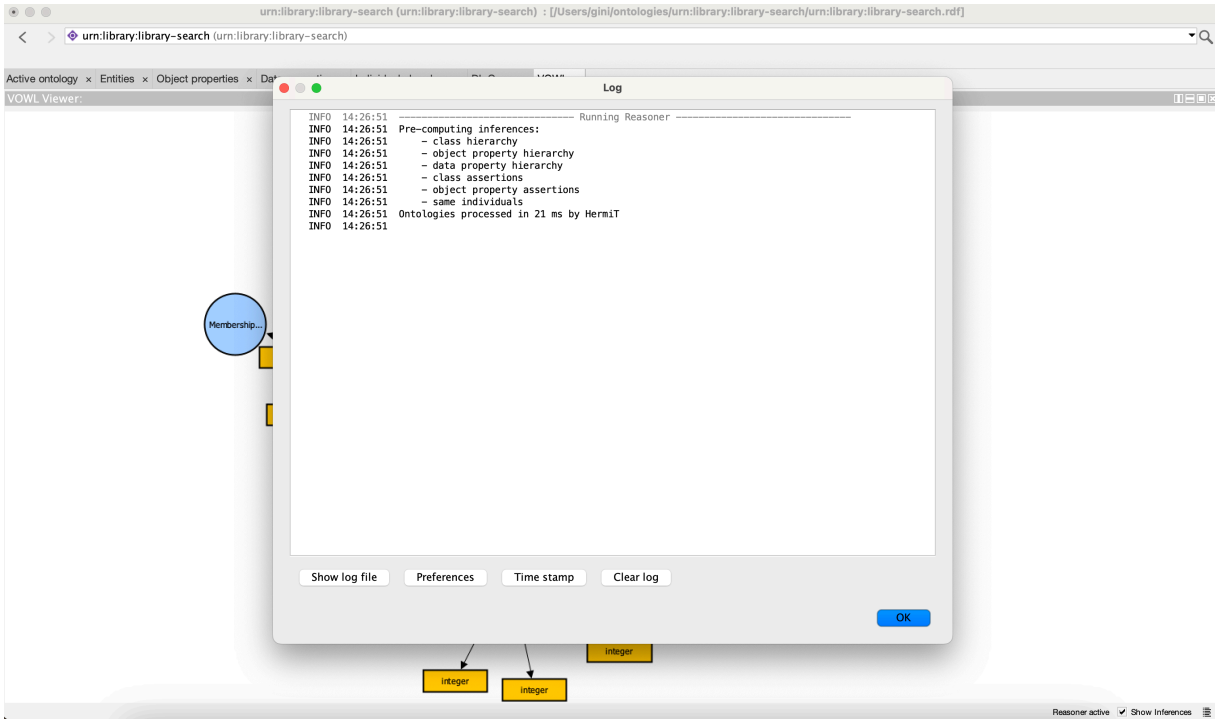
Data Properties



Visual OWL



Reasoner log: HermiT 1.4.3.456



4 Justifying the approach

The approach can be justified for several reasons:

1. Structured and organised knowledge: This makes developing an intelligent search engine to understand and interpret the library's data easier.
2. Standardisation of terms: The search engine can return more accurate and relevant search results by standardising the terms.
3. Faceted search: This makes it easier for users to find the books they are interested in and helps them discover new books they may not have considered otherwise.
4. Personalisation: By linking the user class in ontology to the books they have borrowed, the search engine can recommend similar books that the user may be interested in.
5. Semantic enrichment: This can help improve the search experience by providing more relevant search results and helping users discover new books they may not have considered otherwise.

5 Testing the approach

One way to test the ontology is to use the reasoning feature of Protege to check if the ontology is consistent and follows the rules of the domain. This can be done by running the reasoner on the ontology and examining the inferred axioms to ensure they are correct and make sense.

Another way to test the ontology is to use the query feature of Protege to test the ontology's ability to retrieve information. This involves creating a query using SPARQL or DLQuery.

The ontology can also be tested using Protégé's built-in visualisation tools, such as the class hierarchy view and the property hierarchy view. These tools allow users to examine the ontology's structure and check for any errors or inconsistencies.

Furthermore, Protege allows for creating individuals, which are specific instances of the classes in ontology. By creating individuals and adding them to the ontology, one can test the ontology's ability to reason about specific instances of the classes. This can be useful for testing the ontology's ability to make recommendations based on user borrowing history.

Overall, while testing the ontology prototype in Protege does not provide a complete picture of the search engine's outputs, it can be used to ensure that the ontology is well-formed and accurate.

6 Applying the approach and methods

The approach and methods developed for the ontology prototype can be applied to other problems related to information retrieval and management. By creating an ontology specific to a particular domain, we can improve the accuracy and relevance of search results, leading to a more efficient and effective search experience.

The approach can also be applied to other domains, such as e-commerce. For example, an ontology can be created to categorise products, allowing for more accurate and relevant search results.

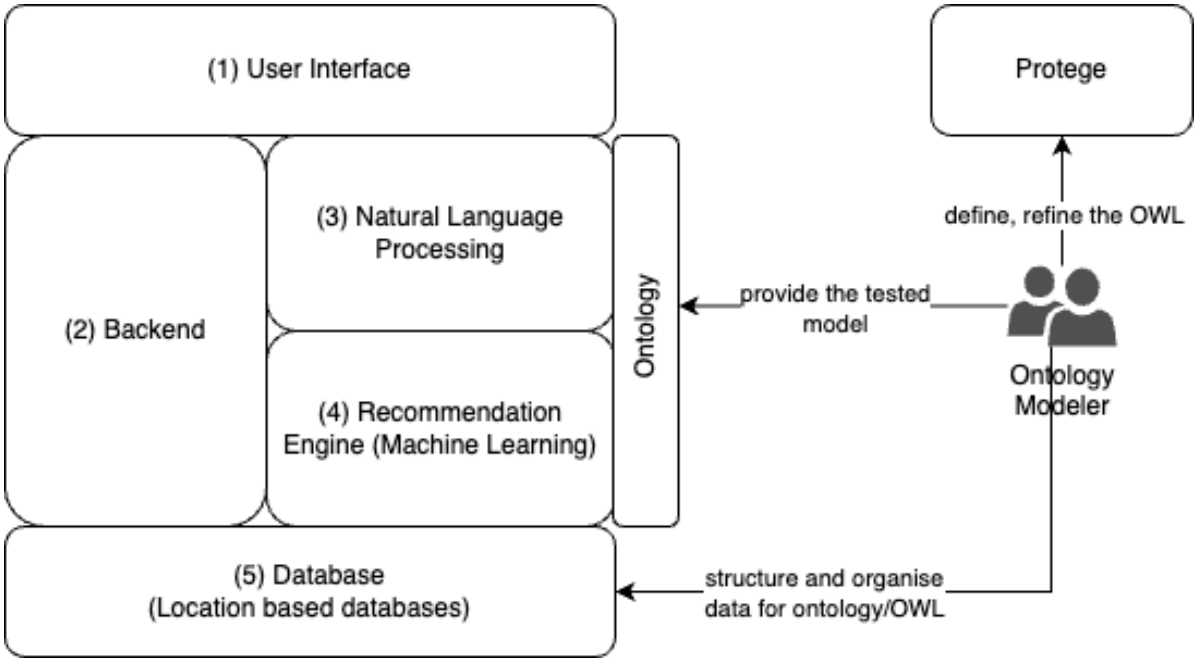
In the healthcare industry, by categorising medical data into classes and properties, it is possible to create a more efficient and effective search experience for healthcare providers, leading to better patient outcomes.

Overall, the approach and methods developed for the ontology prototype can be applied to various problems related to information retrieval and management.

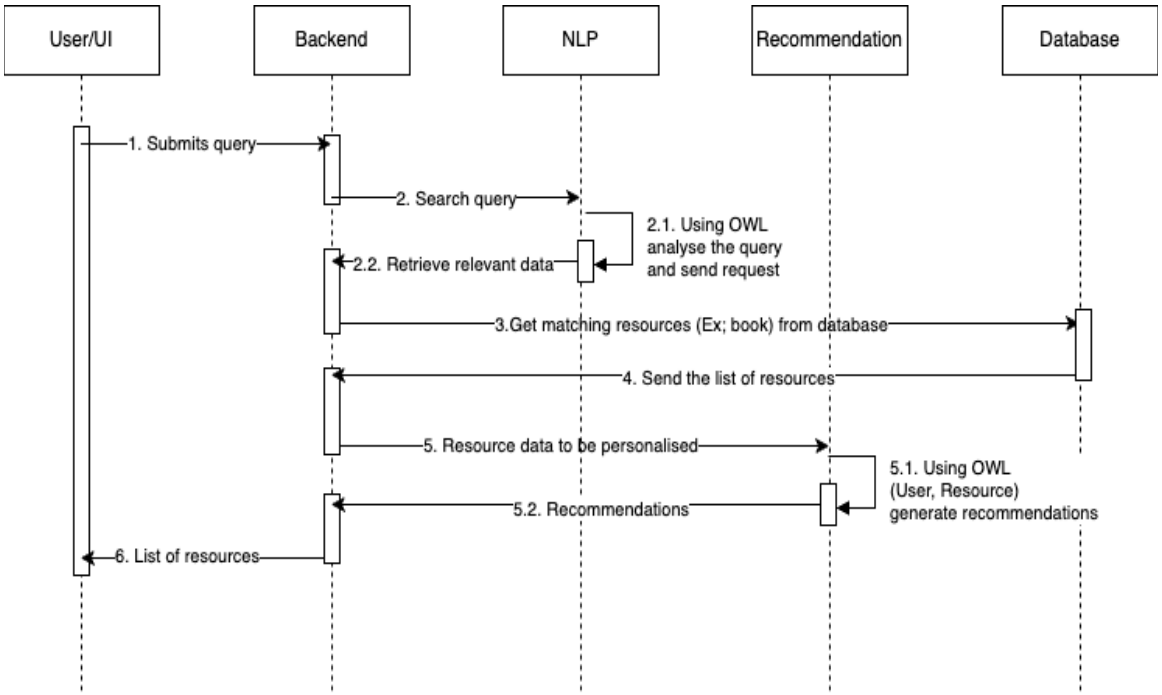
7 Architecture concept – Putting it together

At a high level to get the best out of using the designed ontology would be to use it as part of the architecture that would become the software for the library. (International Journal of Web & Semantic Technology)

A typical architecture would look like this:



Explaining the architecture and how ontology helps in retrieving the data based on the submitted search can be viewed in a sequence diagram:



8 Conclusion

Developing an ontology prototype for a local library search engine using Protégé software has shown how ontology can be used to organise and categorise data, making it easier to search and retrieve.

The approach and methods used to develop the ontology prototype can be applied to other problems related to information retrieval and management in various domains, such as e-commerce and healthcare. Furthermore, by creating an ontology specific to a particular field, we can improve the accuracy and relevance of search results, leading to a more efficient and effective search experience.

Testing the ontology prototype in Protégé allows us to ensure that the ontology is well-formed and accurate.

Overall, developing an ontology prototype for a local library search engine using Protégé software has demonstrated the potential of ontology in information retrieval and management and highlighted the importance of well-structured and organised data in improving search experiences.

9 References

International Journal of Web & Semantic Technology (IJWesT) Vol.2, No.4, October 2011

Knublauch H, Horridge M, Musen M, Rector A, Stevens R, et al. (2005) The Protégé OWL Experience, Workshop on OWL: Experiences and Directions. Fourth International Semantic Web Conference (ISWC2005), Galway, Ireland.

Medinal MA, Sanchezz JA, Callejal JDL, Benitez1 A, A practical approach to model classification schemes with OWL ontologies.

Ontology definition by Tom Gruber:www.tomgruber.org

Protégé tool for ontology: www.w3.org/protege

Semantic web definition: www.w3.org/standards/semanticweb

W3C Semantic Web Activity. World Wide Web Consortium (W3C). November 7, 2011. Retrieved November 26, 2015.

Weronika T. Adrian. Antoni Ligeza. Grzegorz J. Nalepa. Krzysztof Kaczor, Distributed and collaborative knowledge management using an ontology-based system, International Fedration for Information Processing, 2014, pp112-130.

Zhang, Lei, Jing Li (2011) Automatic generation of ontology based on database. Journal of Computational Information Systems 7: 1148-1154.

10 Miscellaneous

Python code using Ontology and NLP to extract concepts and keywords.

The function first parses the query using the NLP model and then iterates over each token in the parsed query. For each token, it checks whether it corresponds to a concept (e.g., "romantic" genre) or a keyword (e.g., "novels") using the ontology. If the token corresponds to a concept, it adds it to the list of concepts. If the token corresponds to a keyword, it searches the ontology for a matching keyword and adds it to the list of keywords.

Finally, the function returns the list of extracted concepts and keywords, which can be used by other components of the book search engine to filter and rank relevant books.

By loading the ontology into the Python environment using **owlready2**, we can use the ontology to query and reason about the domain knowledge related to the library system. The **onto.search_one(label=token.lemma_)** function call in the above code searches for an individual in the ontology with a label matching the lemmatised form of the keyword token. This allows us to identify relevant keywords that are associated with specific books in the library and use them to filter and rank search results.

```
# Import required libraries
from owlready2 import *
import spacy

# Load the ontology
onto_path.append("/path/to/ontology")
onto = get_ontology("library.owl").load()

# Load the NLP model
nlp = spacy.load("en_core_web_sm")

# Define a function to extract concepts and keywords from a search query
def extract_concepts(query):
    # Parse the query using the NLP model
    doc = nlp(query)

    # Extract relevant concepts and keywords using the ontology
    concepts = []
    keywords = []
    for token in doc:
        if token.ent_type_ == "CONCEPT":
            concepts.append(token.text)
        elif token.pos_ in ["NOUN", "ADJ", "VERB"]:
            keyword = onto.search_one(label=token.lemma_)
            if keyword:
                keywords.append(keyword)
```

```
    # Return the extracted concepts and keywords
    return concepts, keywords
```

Python Code retrieving the list of books from a library database based on the search criteria.

The **search_books** function takes a user query as input, which is then parsed using NLP to extract relevant keywords. The function then uses the ontology to identify individuals in the ontology that match the keywords and queries the database for books that contain these keywords in their **keywords** field.

The resulting books are then ranked by relevance based on the number of matching keywords, and the results are displayed to the user. The **Book** class defined using SQLAlchemy provides an ORM mapping between the database and the Python code, allowing us to query and manipulate the data using Python objects easily.

```
from owlready2 import *
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Load the ontology
onto = get_ontology("library.owl").load()

# Connect to the library database
engine = create_engine('sqlite:///library.db')
Session = sessionmaker(bind=engine)
Base = declarative_base()

# Define the Book class for ORM
class Book(Base):
    __tablename__ = 'books'
    id = Column(Integer, primary_key=True)
    title = Column(String)
    author = Column(String)
    genre = Column(String)
    keywords = Column(String)

# Define the search function
def search_books(query):
    # Parse the user query using NLP
```

```
    tokens = nlp(query)
    # Extract relevant keywords using the ontology
    keywords = [token.lemma_ for token in tokens if token.pos_ in ["NOUN", "ADJ",
"VERB"] and onto.search_one(label=token.lemma_)]
    # Query the database for matching books
    session = Session()
    results = session.query(Book).filter(Book.keywords.contains("
".join(keywords))).all()
    session.close()
    # Rank the results by relevance
    ranked_results = sorted(results, key=lambda x: sum([1 for keyword in keywords if
keyword in x.keywords]), reverse=True)
    # Display the results
    for book in ranked_results:
        print(f"{book.title} by {book.author} ({book.genre}): {book.keywords}")
```